Slouching Towards Exascale

Ewing Lusk
Mathematics and Computer Science Division
Argonne National Laboratory

Introduction

Let us speculate about how we will program exascale machines. Some believe that the current "standard" of MPI plus a venerable sequential language (Fortran, C, or C++) will become as abruptly obsolete as the vector Fortran compilers of the 1970s. While it is exciting to contemplate an *ab initio* redesign of the HPC software infrastructure, experience tells us that large-scale software (and HPC software is now very large scale) requires a migration path that consists of incremental steps during which only some parts change at a time. Indeed, as scalability forced vectorization to give way to message passing, Fortran changed a little but was not replaced by Ada.

Where We Are Now

We are about to take another major step, but not a cataclysmic one. We now have robust, portable, and effective standard languages for programming a von Neumann machine with a single program counter and a single address space. Thanks to MPI, we have a robust, portable, and effective standard for communication and synchronization among such machines, What we lack is a robust, portable, and effective standard for parallel programming (multiple program counters) within a single address space. (Neither OpenMP nor POSIX pthreads provide features needed for an approach effective for HPC.)

MPI, admittedly cumbersome for some straightforward tasks, has become the universal mechanism for expressing parallelism among multiple address spaces, for several reasons. Designed through a completely open process, it included the concerns of multiple stakeholders from the beginning. This process resulted in a definition that was portable to a wide class of machines and with a certain degree of performance transparency that encourages the development of high-performance, scalable libraries and applications. MPI's design favored the development of portable libraries over end-application programs, and in this it has been successful. Its specification includes language interoperability and other features that enable it to fit into the HPC ecosystem with existing tools. These properties are worth reviewing because we must be sure that what we add to our programming environment be not *worse* than MPI.

The Next Step

The next step we are about to take is forced upon us by physics, so it is pointless to resist. Because of power and heat dissipation requirements, multicore chips are already with us. Whatever shape exascale computers ultimately take, we will be programming machines with less memory per processing core than we are now. This reality will force most (not all) applications to augment their existing programming model to include parallelism within an address space together with their current MPI-based parallelism across multiple address spaces.

This "hybrid" style of programming is already being used by applications in many areas as they migrate toward petascale. The current most common shared-memory approach is OpenMP. Although high-performance programming is difficult with OpenMP because of its lack of locality control, OpenMP+MPI is virtually the only approach being widely used, for several reasons: (1) OpenMP is available on a wide variety of machines; (2) both Fortran and C are supported; and (3) the OpenMP and MPI standards make explicit commitments to each other that provide clear semantics for various levels of thread safety in hybrid programs. The fact that OpenMPI+MPI represents an incremental step for most applications (the overall MPI structure of the application can be maintained while the MPI processes are internally parallelized with OpenMP threads) is an important factor in encouraging applications to move to a hybrid model.

But OpenMP, at least as currently defined and implemented, is unlikely to be the final answer for shared-memory parallel programming. In addition to the lack of locality control, most implementations are restricted to single-node parallelism, where the hardware provides the shared memory and synchronization mechanisms. Applications are already finding the need for larger memories to be associated with their MPI processes than are hosted on the single nodes of petascale machines. Therefore it may be useful to consider the PGAS languages (UPC, Co-Array Fortran, and Titanium), which offer a shared-memory model with a distinction between local and shared memory, thus providing locality control and performance transparency.

What the PGAS languages lack so far is clear semantics for interaction with MPI and implementations to match. One can imagine a million-thread computation organized as 10,000 UPC or CAF address spaces with 100 threads each, communicating via MPI, which strains the scalability of neither model. Again, this would be an incremental change for an existing MPI application.

Libraries

In discussing approaches to parallel programming, one often forgets that not all programmers require the same features from their programming models. Let us define a *library* as a collection of functions that are usable in multiple applications. Writers of such libraries need access to performance and (except for certain vendor-specific libraries) portability. To obtain these features, they are willing to give up a certain degree of ease of use. Application writers, on the other hand, wish to focus on their science and would rather not cope with some of the details required for scalability and performance. For them, the easier it is to develop applications, the better they can produce computational science results.

We are most familiar with the dichotomy between application and library in the case of mathematical software, since the mathematics is the same for so many applications. But there also exist libraries that are specialized to certain families of algorithms rather than areas of application. For example, researchers have expressed interest in sophisticated load-balancing libraries that can hide all of the MPI communication from an application code, simultaneously providing scalability while simplifying the application logic.

What We Need to Do

Four actions would make progress toward programming exascale machines.

- *Eschew ritualized denigration of MPI.* It is a robust definition, with robust implementations, of a critical component of future programming systems, namely the transfer of data among separate address spaces. Support continued research into areas of MPI that need it. The MPI-3 Forum is at work on extending the standard.
- *Recognize the need for a shared-memory programming model.* What current applications and libraries alike will embrace is a programming system for parallelism within an address space. Such a system needs to be comparable with MPI in portability and performance transparency. It need not be scalable to the ultimate levels, but should not be restricted to running on a single node. Clear semantics for interoperability with MPI are required. This is a critical research topic; multiple solutions should be pursued at this point. PGAS languages show promise, but semantics for interoperability with MPI are not yet there.
- *Understand the difference between end applications and libraries*. While some applications will use hybrid systems consisting of explicit management of parallelism within an address space together with MPI, other applications may be able to rely on libraries, some of them specialized to single algorithms or domains.
- *Don't abandon the HPCS language ideas*. While separate, vendor-sponsored development of multiple "high-productivity" languages has not attracted much attention from application programmers yet, the HPCS languages (Chapel, X10, and Fortress) have introduced a number of important ideas. An open, multi-agency program with a clearly defined research focus could ultimately bear significant fruit.

Conclusion

This has been necessarily a simplified speculation on programming models for exascale machines. In particular, it has largely ignored the issue of GPUs (although they often come with their own shared address space and thus require a shared-memory programming model) and has focused on hierarchies having depth of only two. Even within these simplifications, however, many challenges and exciting research opportunities exist on the path to exascale.

Acknowledgment

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contract DE-AC02-06CH11357.